

UNITED STATES PATENT APPLICATION

FOR

KERNEL CRYPTOGRAPHIC MODULE SIGNATURE
VERIFICATION SYSTEM AND METHOD

INVENTOR:

BHARGAVA K. YENDURI

KERNEL CRYPTOGRAPHIC MODULE SIGNATURE VERIFICATION SYSTEM AND METHOD

FIELD OF THE INVENTION

5 The present claimed invention relates generally to the field of computer operating systems. More particularly, embodiments of the present claimed invention relate to a system for verifying kernel module signature data.

BACKGROUND ART

10

A computer system can be generally divided into four components: the hardware, the operating system, the application programs and the users. The hardware (e.g., central processing unit (CPU), memory and input/output (I/O) devices) provides the basic computing resources. The application programs (e.g.,
15 database systems, games, business programs (database systems, etc.) define the ways in which these resources are used to solve computing problems. The operating system controls and coordinates the use of the hardware resources among the various application programs for the various users. In doing so, one goal of the operating system is to make the computer system convenient to use.
20 A secondary goal is to use the hardware in an efficient manner.

The Unix operating system is one example of an operating system that is currently used by many enterprise computer systems. Unix was designed to be a time-sharing system, with a hierarchical file system, which supported multiple
25 processes. A process is the execution of a program and consists of a pattern of bytes that the CPU interprets as machine instructions (text), data and stack. A

stack defines a set of hardware registers or a reserved amount of main memory that is used for arithmetic calculations.

5 The Unix operating system consists of two separable parts: the "kernel" and the "system programs." Systems programs consist of system libraries, compilers, interpreters, shells and other such programs that provide useful functions to the user. The kernel is the central controlling program that provides basic system facilities. The Unix kernel creates and manages processes, provides functions to access file-systems, and supplies communications facilities.

10

 The Unix kernel is the only part of Unix that a user cannot replace. The kernel also provides the file system, CPU scheduling, memory management and other operating-system functions by responding to "system-calls." Conceptually, the kernel is situated between the hardware and the users. System calls are used
15 by the programmer to communicate with the kernel to extract computer resource information. The robustness of the Unix kernel allows system hardware and software to be dynamically configured to the operating system while applications programs are actively functional without having to shut-down the underlying computer system.

20

 Thus, when system hardware or software resource changes are implemented in a computer system having the Unix operating system, the kernel is typically configured or reconfigured to recognize the changes. These changes are then made available to user applications in the computer system.

25 Furthermore, as system errors and faults occur in the underlying operating system, the kernel is able to identify these errors and faults and make them available to applications that these error and faults may affect. Applications

typically make system calls by way of "system traps" to specific locations in the computer hardware (sometimes called an "interrupt" location or vector) to collect information on these system errors. Specific parameters are passed to the kernel on the stack and the kernel returns with a code in specific registers indicating
5 whether the action required by the system call was successfully completed or not.

Figure 1 is a block diagram illustration of an exemplary prior art computer system 100. The computer system 100 is connected to an external
10 storage device 180 and to an external drive device 120 through which computer programs can be loaded into computer system 100. The external storage device 180 and external drive 120 are connected to the computer system 100 through respective bus lines. The computer system 100 further includes main memory 130 and processor 110. The drive 120 can be a computer program product reader
15 such a floppy disk drive, an optical scanner, a CD-ROM device, etc.

Figure 1 additionally shows memory 130 including a kernel level memory 140. Memory 130 can be virtual memory which is mapped onto physical memory including RAM or a hard drive, for example. The robustness of the
20 Unix operating system and the dynamic ability to reconfigure the Unix kernel also makes the kernel susceptible to unauthorized access and intrusive attacks.

There is currently cryptographic software that only operate in the application space of the operating system of the personal computer. Therefore, it
25 can only be called upon by an application, such E-mail, Microsoft Word or the like. In this prior art system, the cryptographic software cannot work in the kernel space of the operating system. The kernel space is that layer of operating

system which is essentially non-visible to the user at the driver level of the computer system, for example, where IP packets are processed, where a disc drive controller software resides, where the computer system's printer drivers are located, etc.

5

In this prior art embodiment, kernel space routines cannot cross the line into application space. Therefore, if one wants to encrypt data or instructions coming in or out of the hard drive, the cryptographic software would not be usable, as it resides in the application space and not in the kernel space.

10 Similarly, the IP packets would also be unable to be encrypted using cryptographic software as the IP packets are processed in the kernel space.

To enable cryptographic software operations in the kernel space, some prior art systems as illustrated in Figure 2 provide cryptographic software
15 implementation wherein the software is situated in each application space and in devices that access the kernel space of a standard operating system. The separate application space and device space software are then linked together to exchange cryptographic functions, such as algorithms, digital signatures, hash functions, etc.

20

In the this prior art solution, each application space and device space cryptographic software includes a generic layer with a program interface and a cryptographic service module having a library of encryption algorithms which electronically communicate with the program interface. However, this prior art
25 solution does not have a standard authentication and security approach to prevent unauthorized and unwanted applications and devices from intruding he kernel. This prior art solution is also cumbersome and costly since each device

wishing to access the kernel's cryptographic service needs a copy of the
cryptographic software loaded in it. The prior art solution further does not
offers any reliable way of ensuring that only verified devices or device drivers
are accessing the kernel since non-validated kernel modules are able to by-pass
5 the kernel's encryption scheme to write code to the kernel driver layer

Furthermore, the kernel verification methods by current prior art
techniques require massive amounts of redundant data which unnecessarily
consume system resources, particularly memory. The unnecessary consumption
10 of system resources by the large volume of prior art kernel verification processes
also results in performance degradation to the underlying computer system.

15

20

25

SUMMARY OF INVENTION

Accordingly, to take advantage of the many security application programs available and the increasing number of new applications being developed to handle the associated security issues, a system is needed that allows a
5 programmer to add extensions to a kernel to automatically verify kernel module additions to the kernel of kernel cryptographic modules without disrupting the functionality of the kernel for other operations. Further, a need exists to automatically generate kernel module verification data without having to unduly burden system resources in the underlying computer system. A need
10 further exists for an improved and less costly program independent operating system, which improves efficiency, reliability and provides a means to implement kernel level cryptographic verification without losing the embedded features designed in the kernel.

15 What is described herein is a computer system having a kernel structure that provides a technique for automatically providing secure method for conveying the results of signature based kernel module verification from the loading of loadable kernel modules, and ensuring the binding between the signature and the image of the module being loaded. Embodiments of the
20 present invention allow programmers to automatically verify and authenticate kernel modules attempting to load into the kernel to become cryptographic service providers without having to use up substantial portions of kernel memory. Embodiments of the present invention allow kernel modules to be loaded into a kernel using a signature verification scheme that minimizes the
25 potential of unauthorized modules from being loaded into the kernel.

The kernel module authentication system provides loadable kernel modules with an authenticating means of authenticating and verifying kernel modules prior to these modules being loaded into the kernel.

5 Embodiments of the present invention include kernel cryptographic framework daemon that helps in managing signature information validation between the kernel cryptographic framework and external storage device in the underlying computer system. The kernel cryptographic framework daemon also does module verification for kernel cryptographic modules.

10

 Embodiments of the present invention also include kernel cryptographic framework that enables authenticated and verified kernel cryptographic modules to be loaded into the kernel as cryptographic service providers. The kernel cryptographic framework communicates with the kernel cryptographic
15 daemon using input/output control commands to initiate the authentication and verification scheme of the present invention.

 Embodiments of the present invention further include kernel cryptographic verification logic for verifying the kernel module signatures to
20 ensure that files whose signatures are verified are the one being loaded and installed in the kernel. The verification module includes logic to store the signature information and signature length of each file in order to enable the verification of signature information of each file.

25 Embodiments of the kernel cryptographic framework of the present invention include registration logic that allows each kernel cryptographic module to register with the kernel cryptographic framework for verification and

authentication. The registration logic enables the kernel cryptographic framework to compare signature information presented by a particular kernel module with the signature information retrieved by the kernel cryptographic framework daemon in order to determine the security of the signature presented.

5

Embodiments of the kernel cryptographic framework of the present invention include a framework communication routine that allows the kernel cryptographic framework to communicate with the kernel cryptographic framework daemon. The kernel cryptographic framework and daemon interact using input/output control commands. The daemon calls these commands during an initial communication routine of the kernel cryptographic framework verification scheme and during the operation of the kernel cryptographic framework verification process of the present invention.

15 These and other objects and advantages of the present invention will no doubt become obvious to those of ordinary skill in the art after having read the following detailed description of the preferred embodiments which are illustrated in the various drawing figures.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

5

Figure 1 is a block diagram of a prior art computer system;

Figure 2 is a block diagram of a prior art kernel cryptographic verification method of a prior art computer system;

10

Figure 3 is a block diagram of an embodiment of a computer system in accordance with the present invention;

15

Figure 4 is a block diagram of one embodiment of the kernel module verification system of one embodiment of the kernel data formatting system of the present invention;

20

Figure 5 is a block diagram of one embodiment of an internal architecture of one embodiment of the kernel cryptographic framework of one embodiment of the present invention; and

25

Figure 6 is a flow diagram of one embodiment of an exemplary module verification interaction of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings.

While the invention will be described in conjunction with the preferred

5 embodiments, it will be understood that they are not intended to limit the invention to these embodiments.

On the contrary, the invention is intended to cover alternatives,

modifications and equivalents, which may be included within the spirit and

10 scope of the invention as defined by the appended Claims. Furthermore, in the following detailed description of the present invention, numerous specific details are set forth in order to provide a thorough understanding of the present

invention. However, it will be obvious to one of ordinary skill in the art that the present invention may be practiced without these specific details. In other

15 instances, well-known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

The embodiments of the invention are directed to a system, an

20 architecture, subsystem and method to automatically verify and authenticate kernel cryptographic modules in an underlying kernel space in a computer system. In accordance with an aspect of the invention, a kernel module signature information is processed and verified to protect the security of the computer system from unauthorized system intruders to the underlying

25 operating system and the other applications running in the computer system.

Figure 3 is a block diagram illustration of one embodiment of a computer system 300 of the present invention. The computer system 300 according to the present invention is connected to an external storage device 380 and to an external drive device 320 through which computer programs according to the present invention can be loaded into computer system 200. External storage device 380 and external drive 320 are connected to the computer system 200 through respective bus lines. Computer system 300 further includes main memory 330 and processor 310. Drive 320 can be a computer program product reader such a floppy disk drive, an optical scanner, a CD-ROM device, etc.

10

Figure 3 shows memory 330 including a kernel level memory 340. Memory 330 can be virtual memory which is mapped onto physical memory including RAM or a hard drive, for example, without limitation. During process execution, a programmer programs data structures in the memory at the kernel level memory 340.

15

According an embodiment of the present invention, the kernel memory level 340 includes a kernel module signature verification system (KMSVM) 350. The KMSVM 350 comprises kernel cryptographic framework (KCF) and a kernel cryptographic framework daemon (KCFD) for processing kernel cryptographic module signatures to authenticate and verify the kernel modules ability to be loaded into the kernel memory 340.

20

The KMSVM 350 automatically monitors kernel module path and extracts the signature information provided by each module attempting to load to the kernel 340. The signature information retrieved for the kernel module path is provided by the kernel cryptographic framework daemon to verify the signature

25

information provided by the kernel cryptographic framework when the same kernel module attempts to register the signature information with the kernel cryptographic framework.

5 Figure 4 is a block diagram illustrating an implementation of one embodiment of the KMSVM 350 of the present invention. The KMSVM 350 comprises kernel cryptographic framework daemon 400, kernel cryptographic framework 410, kernel subsystems 420 - 430, kernel cryptographic modules 440 - 460 and kernel cryptographic module signatures 441 - 461.

10

 The kernel cryptographic framework daemon (KCFD) 400 is a daemon which helps in retrieving signature verification information from a storage device in the computer system every time a cryptographic module attempts to request to be loaded into the kernel 340. In one embodiment of the present
15 invention, the KCFD 400 does module verification for kernel cryptographic modules 440 - 460 using information extracted from the file path names of the modules 440 - 460. In one embodiment of the present invention, the KCFD 400 interacts with the kernel cryptographic framework (KCF) 410 using input/output control commands.

20

 In one embodiment of the present invention, when a cryptographic module requests to be loaded into the kernel 340, the module path information presented by the module to the KCF 410 is passed on to the KCFD 400 which retrieves a corresponding signature information from system storage. The KCFD
25 400 responds to the verification request by providing the KCF 410 signature status information, signature length information and signature size information.

An exemplary signature verification communication code between the KCFD 400 and the KCF 410 is as follows:

```

5      #ifdef_cplusplus
      extern "C" {
      #endif
      typedef enum ELFSign_status_e {
          ELFSIGN_UNKNOWN,
          ELFSIGN_SUCCESS,
10         ELFSIGN_FAILED,
          ELFSIGN_NOTSIGNED
      } ELFSign_status_t;

      #define KCF_KCFD_VERSION 1 1
15     #define SIG_MAX_LENGTH    1024

      typedef struct kcf_door_arg_s {
          short  da_version;

20         union {
            char filename [MAXPATHLEN]; /*For request */
            struct kcf_door_results_s { /*For response*/
                ELFSign_status_t      status;
                unit32_t               siglen;
25                uchar_t             signature [SIG_MAX_LENGTH];
            } result;
        } da_u
      } kcf_door_arg_t;
      #ifdef_cplusplus
30     }
      #endif

```

Reference is now made to Figure 5 which is a block diagram illustration of one embodiment of an internal architecture of the KCF 410 of the present invention. As depicted in Figure 4, the KCF 410 comprises communication module 500, registration module 510, verification module 520 and data structure module 530.

The communication module 500 enables the KCF 410 to communicate with the KCFD 400. In one embodiment of the present invention, the KCF 410 and the KCFD 400 communicate using input/output control commands that allows the KCF 410 to generate doors that are created by the KCFD 400 during system boot time and a door ID is passed to the kernel 340. Since there isn't a door file, it is difficult for an intruder to snoop on communications between the KCFD 400 and the KCF 410. In one embodiment of the present invention, the communication module 500 is implemented as system calls that the KCF 410 may make to the kernel to pass data to and from the KCFD 400.

10

The registration module 510 enables the KCF 410 handle load registration requests by cryptographic modules wishing to join the kernel as cryptographic service providers in the kernel space. In one embodiment of the present invention, when cryptographic modules register, they have to be authenticated prior to being added to the list of service providers by the KCF 410.

15

As part of the registration process the KCF 410 keeps the signature information and signature length data for each requesting cryptographic module in its internal data structure 530. The information retained in data structure 530 is compared with signature information received from the KCFD 400 in response to KCF 410 requests.

20

Each signature information provided by a requesting cryptographic module is verified by the verification module 520 in order to determine whether the requesting module is a legitimate module that must be loaded by the KCF 410. In one embodiment of the present invention, signature information provided by each requesting cryptographic module is the result of compiling,

25

among other cryptographic measures, an internal private security key and a public security key. In one embodiment of the present invention, additional precautionary security measures may be adopted to ensure that only those modules authenticated and verified are allowed to be loaded to the list of service
5 providers. In one embodiment of the present invention, the structure module 530 may be extended to add additional fields to further restrict the ability of intruding modules to be loaded by the KCF 410.

Figure 6 is a computer implemented software of a block diagram
10 illustration of an exemplary computer software implemented interaction pattern 600 of one embodiment of the present invention. As shown in Figure 6, the cryptographic module signature verification process begins at step 605. And at step 610, during system boot time, the KCFD 400 creates an unnamed door and passes the door identification to the KCF 410 by an ioctl command (e.g., (fd,
15 CRYPTO_LOAD_DOOR, ..)). This door is used for communication between the KCFD 400 and the KCF 410. Since there is no door file, it is difficult for an intruder to snoop on this communication.

At step 615, a cryptographic module is loaded and calls the register
20 routine to register with the KCF 410 as a cryptographic provider. In one embodiment of the present invention, the cryptographic module includes a signature section, which is provided to the KCF 410 during the module load. The KCF 410 needs to verify the module before adding it to its list of providers. The KCF 410 does this by making a door upcall and passing the full pathname of
25 the module to the KCFD 400. In one embodiment of the present invention, the register routine is called from an initialization routine for a software provider or from a hardware routine of the hardware provider.

At step 620, the KCFD 400 opens the file/module and calls a verification routine (e.g., (fverify_elf_signature(int fd)) and a get signature routine (e.g., get_elf_signature(char *pathname, char *signature, int *sig_len).

5

At step 625, the KCFD 400 passes of the result of the signature verification of the cryptographic module back to the kCF 410. This result has three components: PASS/FAIL result from the verification; the signature information and the signature length. The signature information and the signature length are also passed to the KCF 410 to address an intrusion scenario.

10

At step 630, the KCF 410 checks the PASS/FAIL result from the KCFD 400 verification of the module signature. And at step 635, if the signature verification result is a "PASS", the signature information and the signature length are compared with corresponding fields in the module structure at step 640.

15

At step 645, if the signature information and signature length provided by the KCFD 400 are identical to the fields in the module structure, the module is determined to have passed verification and is added to the KCF 410 list of cryptographic providers at step 650.

20

At step 655, if the signature information and signature length do not compare with the corresponding fields in the module structure or if the signature verification is a failure from the KCFD 400, the KCF 410 logs a system log warning message and returns from the module registration routine and the cryptographic module is not added to the list of cryptographic providers by the KCF 410.

25

The foregoing descriptions of specific embodiments of the present invention have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit the invention to the precise forms disclosed, and obviously many modifications and variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the Claims appended hereto and their equivalents.